**Numpy**

* Create a 3\*3 array using list\_1 = [1,2,3] list\_2 = [4,5,6] list\_3 = [7,8,9]

Ans->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0]

list\_2 = input\_list[1]

list\_3 = input\_list[2]

import numpy as np

/\* Write your code \*/

* **ADVANTAGES of Numpy**

What is the use of arrays over lists, specifically for data analysis? Putting crudely, it is convenience and speed :

1. You can write vectorised code on NumPy arrays, not on lists, which is convenient to read and write, and concise.
2. NumPy is much faster than the standard Python ways to do computations.

Some reasons for such difference in speed are:

* NumPy is written in C, which is basically being executed behind the scenes
* NumPy arrays are more compact than lists, i.e. they take much lesser storage space than lists
* <https://stackoverflow.com/questions/993984/what-are-the-advantages-of-numpy-over-regular-python-lists>
* <https://stackoverflow.com/questions/993984/what-are-the-advantages-of-numpy-over-regular-python-lists>

Vectorised code typically does not contain explicit looping and indexing etc. (all of this happens behind the scenes, in precompiled C-code), and thus it is much more concise.

* Perform an element-wise multiplication using list\_1 = [2,3,4,5] list\_2 = [7,8,9,6] and obtain the output as a list.  
  Hint: Convert the list to an array and after multiplication convert it back to a list.

Ans ->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0]

list\_2 = input\_list[1]

import numpy as np

/\* Write your code \*/

* **Creating NumPy Arrays :**

There are multiple ways to create NumPy arrays, the most common ones being:

* Convert lists or tuples to arrays using, np.array() as done above
* Initialise arrays of fixed size (when the size is known)

The other common way is to initialise arrays. You do this when you know the size of the array beforehand.

The following ways are commonly used:

* np.ones(): Create an array of 1s
* np.zeros(): Create an array of 0s
* np.random.random(): Create an array of random numbers between 0 and 1
* np.arange(): Create an array with increments of a fixed step size
* np.linspace(): Create an array of fixed length
* np.full(): Create a constant array of any number ‘n’
* np.tile(): Create a new array — by repeating an existing array — for a particular number of times
* np.eye(): Create an identity matrix of any dimension
* np.randint(): Create a random array of integers within a particular range
* Given an integer 'x', create an array of size m\*n having all integer values equal to 'x'.   
  Hint: Use dtype to specify integer.  
    
  Format:  
  Input:  
  Line 1: A single integer 'x'  
  Line 2: A single integer 'm' indicating the number of rows  
  Line 3: A single integer 'n' indicating the number of columns  
  Output: An array of size 'm\*n' having all the values as 'x'  
    
  Example:  
  Input 1:  
  1  
  3  
  3  
  Output 1:  
  [[1 1 1]  
  [1 1 1]  
  [1 1 1]]

Ans ->

# Read the input from stdin()

import sys

lines = sys.stdin.readlines()

int\_x = int(lines[0])

rows\_m = int(lines[1])

cols\_n = int(lines[2])

import numpy as np

/\* Write your code \*/

* Create an array of first 10 multiples of 5 using the 'arange' function.

Ans ->

/\* Write your code \*/

* Given an even integer ‘n’, create an ‘n\*n’ checkerboard matrix with the values 0 and 1, using the tile function.

Format:  
Input: A single even integer 'n'.  
Output: An 'n\*n' NumPy array in checkerboard format.  
  
Example:   
Input 1:   
2   
Output 1:   
[[0 1]   
[1 0]]   
Input 2:   
4   
Output 2:   
[[0 1 0 1]  
[1 0 1 0]  
[0 1 0 1]  
[1 0 1 0]]

Ans->

# Read the variable from STDIN

import numpy as np

n = int(input())

a = np.array([[0,1],[1,0]])

/\* Write your code \*/

# Structure and Content of Arrays :

It is helpful to inspect the structure of NumPy arrays, especially while working with large arrays. Some attributes of NumPy arrays are:

* shape: Shape of array (n x m)
* dtype: data type (int, float etc.)
* ndim: Number of dimensions (or axes)
* itemsize: Memory used by each array element in bytes
* Create an array using list list\_1 = [10,11,12,13] and list\_2 = [15,12,13,14] and print the shape and dimension of the array created.

Ans ->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0]

list\_2 = input\_list[1]

import numpy as np

/\* Write your code \*/

* Predict o/p:

Note : For one-dimensional arrays, indexing, slicing etc. is similar to python lists - indexing starts at 0

* + Given an array array\_1 = [1 2 3 5 4 6 7 8 5 3 2], what will be the output of print(array\_1[:3])
  + print(array\_1[0::2])
* From a 2D array extract all the rows of the 2 column.  
  Hint: 2 column will have index value as 1.

Ans->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

import numpy as np

array\_2d =np.array(input\_list)

/\* Write your code \*/

* Extract all the border rows and columns from a 2-D array.  
    
  Format:  
  Input: A 2-D Python list  
  Output: Four NumPy arrays - First column of the input array, first row of the input array, last column of the input array, last row of the input array respectively.  
    
  Example:  
  Input 1:  
  [[11 12 13 14]  
  [21 22 23 24]  
  [31 32 33 34]]  
  Output 1:  
  [11 21 31]  
  [11 12 13 14]  
  [14 24 34]  
  [31 32 33 34]

Ans->

# Read the input list

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

import numpy as np

# Convert the input list to a NumPy array

array\_2d =np.array(input\_list)

# Extract the first column, first row, last column and last row respectively using

# appropriate indexing

/\* Write your code \*/

* Stacking is done using the and np.vstack()methods.
  + np.hstack() : For horizontal stacking, the number of rows should be the same
  + np.vstack() : For vertical stacking, the number of columns should be the same.
* Horizontally stack two arrays using hstack, and finally, vertically stack the resultant array with the third array.   
    
  Example:  
  Input 1:  
  [[1, 2],  
  [5, 6]]  
    
  [[3, 4],  
  [7, 8]]  
    
  [[9, 10, 11, 12]]  
  Output 1:  
  [[1 2 3 4]  
  [5 6 7 8]  
  [9 10 11 12]]

Ans->

#Read the input

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0]

list\_2 = input\_list[1]

list\_3 = input\_list[2]

# Import NumPy

import numpy as np

/\* Write your code \*/

* Given an array, 'array\_3' divide each element with 5.  
  Hint: Create a vectorized function, then apply it to the array\_3

input\_list = [[1,2,3,4],[4,7,5,6],[9,0,7,8],[6,7,8,5]]

Ans->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0:2]

list\_2 = input\_list[2:4]

import numpy as np

/\* Write your code \*/

* NumPy provides the np.linalg package to apply common linear algebra operations, such as:
* np.linalg.inv: Inverse of a matrix
* np.linalg.det: Determinant of a matrix
* np.linalg.eig: Eigenvalues and eigenvectors of a matrix
* np.dot(a, b) : multiply matrices
* Find the inverse, eigenvalues, eigenvectors, determinants of a given matrix

'array\_1'=[[1,2,3],[2,3,4],[4,2,6]]

Ans->

import ast,sys

input\_str = sys.stdin.read()

input\_list = ast.literal\_eval(input\_str)

list\_1 = input\_list[0]

list\_2 = input\_list[1]

list\_3 = input\_list[2]

import numpy as np

/\* Write your code \*/

**Pandas**

* Python Series and Dataframe :
  + <https://stackoverflow.com/questions/26047209/what-is-the-difference-between-a-pandas-series-and-a-single-column-dataframe>
  + <https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-pandas-series-and-python-lists/27373/2>
  + <https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python>
* A series is similar to a 1-D numpy array, and contains scalar values of the same type (numeric, character, datetime etc.). A dataframe is simply a table where each column is a pandas series.
  + pd.Series() :create pandas series from array-like objects
* Dataframe is the most widely used data-structure in data analysis. It is a table with rows and columns, with rows having an index and columns having meaningful names.
* Create a series using list = [6,7,8,9,2,3,4,5] and print the output series as the square of each number in the list.  
  Hint: If input series = 1,2,3 the output series should be 1,4,9  
  Hint: First create the series and then using apply and lambda find the output series.

Ans->

import numpy as np

import pandas as pd

/\* Write your code \*/

Given a dataframe 'df' use the following commands and analyse the result.

df.describe( )

df.columns( )

df.shape( )

Ans ->

import numpy as np

import pandas as pd

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* Create a panda series that contains the first ‘n’ natural numbers and their respective squares. The first ‘n’ numbers should appear in the index position.   
  Hint: Use manual indexing.  
    
  Format:  
  Input: A natural number 'n'  
  Output: A pandas series with the first 'n' natural numbers in the index position and their respective squares in the adjacent column.  
    
  Example:   
  Input 1:   
  4   
  Output 1:   
  1 1  
  2 4  
  3 9  
  4 16  
  dtype: int64

Ans->

# Read the variable from STDIN

n = int(input())

import numpy as np

import pandas as pd

np\_arr=np.arange(1,n+1)

/\* Write your code \*/

* An important concept in Pandas dataframes is that of row indices. By default, each row is assigned indices starting from 0, and are represented at the left side of the dataframe.
* Using set\_index command set the column 'X' as the index of the dataset and then print the head of the dataset.  
  Hint: Use inplace= False

Ans->

import pandas as pd

df = pd.read\_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')

/\* Write your code \*/

* Sort the dataframe on 'month' and 'day' in ascending order in the dataframe 'df'.

Ans->

import pandas as pd

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

* Print only the even numbers of rows of the dataframe 'df'.  
  Note: Don't include the row indexed zero.

Ans->

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* Print out the columns 'month', 'day', 'temp', 'area' from the dataframe 'df'.

Ans->

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* There are two main ways of indexing dataframes:
  + df.iloc: Position based indexing
  + df.loc: Label based indexing
    1. A single label, e.g.'3'or'row\_index'
    2. A list or array of labels, e.g.['3', '7', '8']
    3. A range of labels, where row\_x and row\_y both are included, i.e.'row\_x':'row\_y'
* Using iloc index the dataframe to print all the rows of the columns at index 3,4,5 .

Ans->

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* Using loc function print out all the columns and rows from 2 to 20 of the 'df' dataset.

Ans->

df = pd.read\_csv('https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF')

/\* Write your code \*/

* Print all the columns and the rows where 'area' is greater than 0, 'wind' is greater than 1 and the 'temp' is greater than 15.

Ans->

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* pd.merge(): Merge multiple dataframes using common columns/keys

pd.concat(): Concatenate dataframes

* + It is used when you have dataframes having the same columns and want to append them (pile one on top of the other), or having the same rows and want to append them side-by-side.
* Perform an inner merge on two data frames df\_1 and df\_2 on 'unique\_id' and print the combined dataframe.

Ans->

import pandas as pd

df\_1 = pd.read\_csv('<https://query.data.world/s/vv3snq28bp0TJq2ggCdxGOghEQKPZo>')

df\_2 = pd.read\_csv('<https://query.data.world/s/9wVKjNT0yiRc3YbVJaiI8a6HGl2d74>')

/\* Write your code \*/

* Append two datasets df\_1 and df\_2, and print the combined dataframe.

Ans->

df\_1 = pd.read\_csv('<https://query.data.world/s/vv3snq28bp0TJq2ggCdxGOghEQKPZo>')

df\_2 = pd.read\_csv('<https://query.data.world/s/9wVKjNT0yiRc3YbVJaiI8a6HGl2d74>')

/\* Write your code \*/

* Given three data frames containing the number of gold, silver, and bronze Olympic medals won by some countries, determine the total number of medals won by each country.

Note:

All the three data frames don’t have all the same countries. So, ensure you use the ‘fill\_value’ argument (set it to zero), to avoid getting NaN values. Also, ensure you sort the final dataframe, according to the total medal count in descending order.

Ans->

import numpy as np

import pandas as pd

# Defining the three dataframes indicating the gold, silver, and bronze medal counts of different #countries

gold = pd.DataFrame({'Country': ['USA', 'France', 'Russia'],

'Medals': [15, 13, 9]})

silver = pd.DataFrame({'Country': ['USA', 'Germany', 'Russia'],

'Medals': [29, 20, 16]})

bronze = pd.DataFrame({'Country': ['France', 'USA', 'UK'],

'Medals': [40, 28, 27]})

/\* Write your code \*/

* Grouping and Summarizing Dataframes
* Splitting the data into groups (e.g. groups of customer segments, product categories, etc.)
* Applying a function to each group (e.g. mean or total sales of each customer segment)
* Combining the results into a data structure showing the summary statistics
* Group the data 'df' by 'month' and 'day' and find the mean value for column 'rain' and 'wind'.

Use df.groupby( )

Ans->

import pandas as pd

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* Columns which are created by the user are known as 'Derived Variables'. Derived variables increase the information conveyed by the dataframe.
* Create a new column 'XY' which consist of values obtained from multiplying column 'X' and column 'Y'.

Ans->

import pandas as pd

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* Group the data 'df' by 'month' and 'day' and find the mean value for column 'rain' and 'wind' using the pivot table command.

Ans->

df = pd.read\_csv('<https://query.data.world/s/vBDCsoHCytUSLKkLvq851k2b8JOCkF>')

/\* Write your code \*/

* In Python, missing data is represented using either of the two objects NaN (Not a Number) or NULL .
  + There are four main methods to identify and treat missing data:
  + isnull(): Indicates presence of missing values, returns a boolean
  + notnull(): Opposite of isnull(), returns a boolean
  + dropna(): Drops the missing values from a data frame and returns the rest
  + fillna(): Fills (or imputes) the missing values by a specified value
* Print out the number of missing values in each column in the given dataframe.

Ans->

import pandas as pd

df = pd.read\_csv('<https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0>')

/\* Write your code \*/

## Treating Missing Values

There are broadly two ways to treat missing values:

1. Delete: Delete the missing values
2. Impute:
   * Imputing by a simple statistic: Replace the missing values by another value, commonly the mean, median, mode etc.
   * Predictive techniques: Use statistical models such as k-NN, SVM etc. to predict and impute missing values

* Find out the percentage of missing values in each column in the given dataset.

Ans->

df = pd.read\_csv('<https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0>')

#Round off percentage values to 2 decimial places.

/\* Write your code \*/

* Remove the missing values from the rows having greater than 5 missing values and then print the percentage of missing values in each column.

Ans->

df = pd.read\_csv('<https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0>')

/\* Write your code \*/

* Impute the mean value at all the missing values of the column 'Product\_Base\_Margin' and then print the percentage of missing values in each column.

Ans->

df = pd.read\_csv('<https://query.data.world/s/Hfu_PsEuD1Z_yJHmGaxWTxvkz7W_b0>')

/\* Write your code \*/

* Predict o/p :
* Consider an (11,12) shape array. What is the index (x,y) of the 100th element?

Note: For counting the elements go row-wise. For example, the array,

Example : [[1, 5, 9],

[3, 0, 2]]

the 5th element would be '0'.

* + Which of the following would extract all the rows of the first 3 columns in a given numpy 2D array ‘a’?
* Given m and n, swap the mth and nth rows of the 2-D NumPy array given below.  
    
  a = [[4 3 1]   
   [5 7 0]   
   [9 9 3]   
   [8 2 4]]   
    
  Example:   
  Input 1:   
  0   
  2   
  Output 1:   
  [[9 9 3]  
  [5 7 0]  
  [4 3 1]  
  [8 2 4]]

Ans->

import numpy as np

# Given array

a = np.array([[4, 3, 1], [5, 7, 0], [9, 9, 3], [8, 2, 4]])

# Read the values of m and n

import sys

lines = sys.stdin.readlines()

m = int(lines[0])

n = int(lines[1])

/\* Write your code \*

* Given a single integer n, create an (n x n) 2D array with 1 on the border and 0 on the inside.   
    
  Note: Make sure the array is of type int.  
    
  Example:   
  Input 1:   
  4   
  Output 1:   
  [[1 1 1 1]   
  [1 0 0 1]  
  [1 0 0 1]  
  [1 1 1 1]]  
  Input 2:   
  2   
  Output 2:   
  [[1 1]  
  [1 1]]

Ans->

/\* Write your code \*